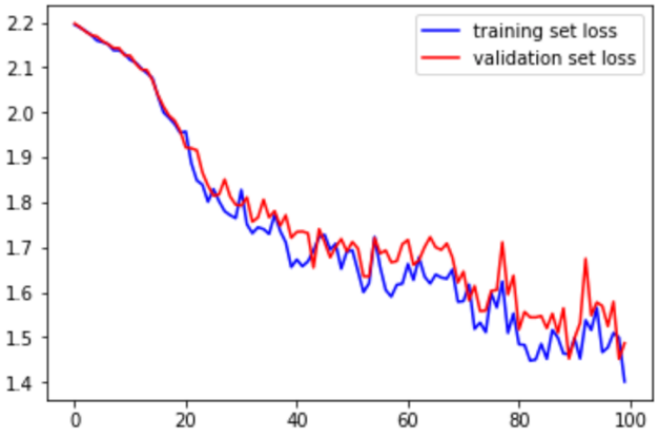
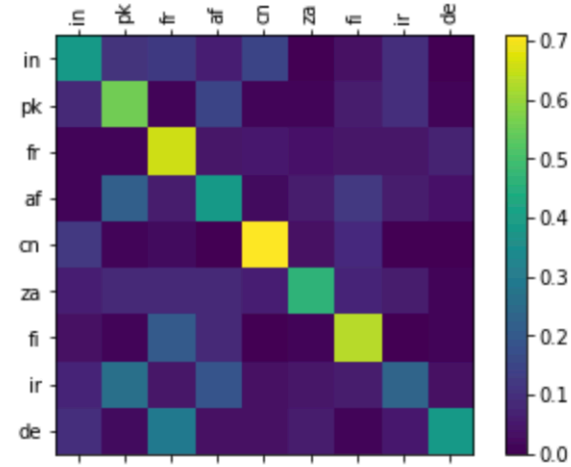
Part 1

1. **Write code to output accuracy on the validation set. Include your best validation accuracy in the report. Discuss where your model is making mistakes and use a confusion matrix plot to support your answer.**

   The best model I have is learning rate = 0.0015, n_hidden = 128, and n_epochs = 100000.





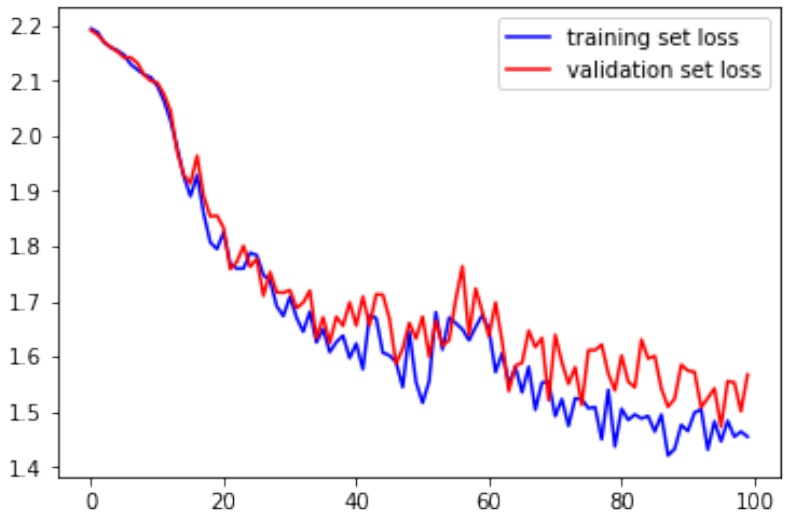
Macro Accuracy: 0.488864
Micro Accuracy: 0.487000

The Macro Accuracy is 0.488864 and the Micro Accuracy is 0.487000.

The left graph shows that the losses from both training set and validation set decrease as number of epoch increases, and the loss of validation set always follows the same trend as the training set. This shows that there is **no overfitting** problem in this model.

From the confusion matrix, we can know that city names from “cn” (China) can be best predicted by the model. The names from “de” (Germany) can be easily classified as from “fi” (Finland). The names from “ir” (Iran) can be eaily classified as from “pk” (Pakistan). Since both Germany and Finland are in Europe and both Iran and Pakistan are in Middle East, these make sense.

2. **Modify the training loop to periodically compute the loss on the validation set, and create a plot with the training and validation loss as training progresses. Is your model overfitting? Include the plot in your report.**



By changing the name dataset to our training sets and validation sets, setting learning rate to 0.002, and keeping all other model parameters the same as the original code (n_hidden = 128, n_epochs = 100000, print_every = 5000, plot_every = 1000), we get graph for the losses of training set and validation set.

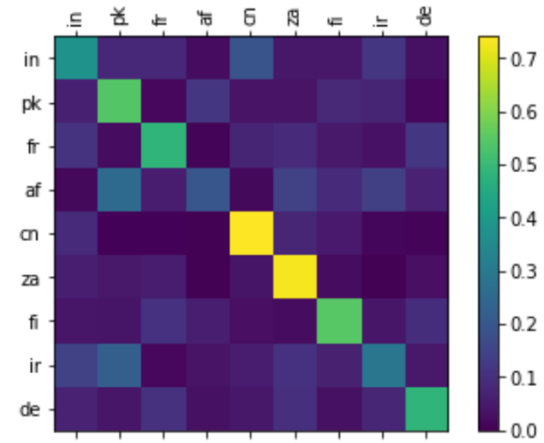
The above graph shows that the losses from both training set and validation set decrease as number of epoch increases. For most of time, the loss of validation set is slightly above the loss of training set but it always follows the same trend as the training set. This shows that there is **no overfitting** problem in my model.

The following are some the loop output:

```
5000 5% (0m 14s) 2.1111 el valle de palomas / fr ✗ (de)
10000 10% (0m 27s) 2.2754 lyulintsy / fi ✗ (in)
15000 15% (0m 43s) 2.9191 cachiriavinha / in ✗ (ir)
20000 20% (0m 57s) 1.8773 qaleh nowye safiabad / af ✗ (ir)
25000 25% (1m 12s) 2.1727 zahinos / cn ✗ (za)
30000 30% (1m 27s) 1.4820 goth jam / af ✗ (pk)
35000 35% (1m 42s) 1.1841 bancos del boral / de ✓
40000 40% (1m 57s) 3.7080 cirojomporang  / fr ✗ (ir)
45000 45% (2m 12s) 0.5611 amiza / za ✓
50000 50% (2m 27s) 1.3730 vuotinainen / cn ✗ (in)
```

```
55000 55% (2m 45s) 0.7293 chauverneneyre / fr ✓
60000 60% (3m 0s) 1.9626 zard ab / pk ✗ (ir)
65000 65% (3m 15s) 1.8875 kairovo / fi ✗ (ir)
70000 70% (3m 31s) 1.3117 kabarwala / pk ✓
75000 75% (3m 47s) 0.6135 basti ajmal khan / pk ✓
80000 80% (4m 2s) 0.9758 pirlaks / fi ✓
85000 85% (4m 17s) 2.2628 kafer nasseh / de ✗ (af)
90000 90% (4m 32s) 3.0838 ballipadara / fi ✗ (in)
95000 95% (4m 46s) 0.6821 hirmula / fi ✓
100000 100% (5m 1s) 0.2745 saintjuliendechedon / de ✓
```

The accuracy and confusion matrix of this model based on validation data are:



Macro Accuracy: 0.488409
Micro Accuracy: 0.487100

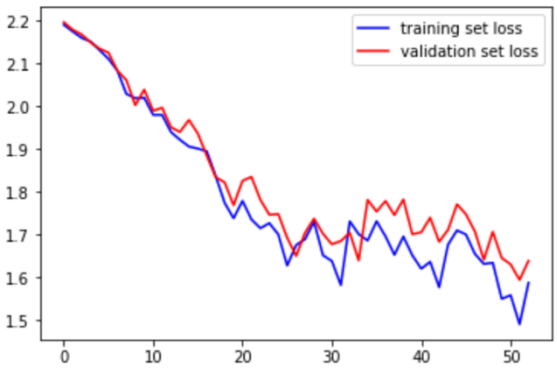
The accuracy is below 50%., so we need to build better model later.

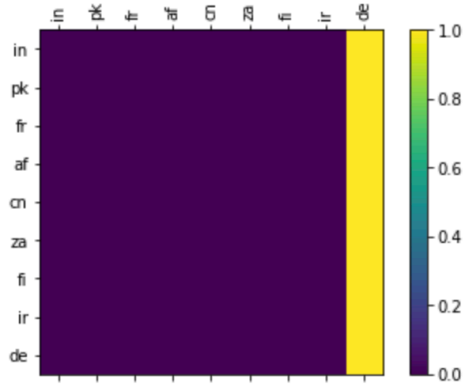
Note: The output you see from the jupyter notebook may not be the same as here, because I run the code for several times, but they are similar.

3. Experiment with the learning rate (at least 5 different learning rates). You can try a few different learning rates and observe how this affects the loss. Use plots to explain your experiments and their effects on the loss.

By keeping all other parameters as the same as default (n_hidden = 128, n_epochs = 100000, print_every = 5000, plot_every = 1000, etc), I try 5 different learning rates, 0.003, 0.002, 0.0015, 0.0001, and 0.0005.

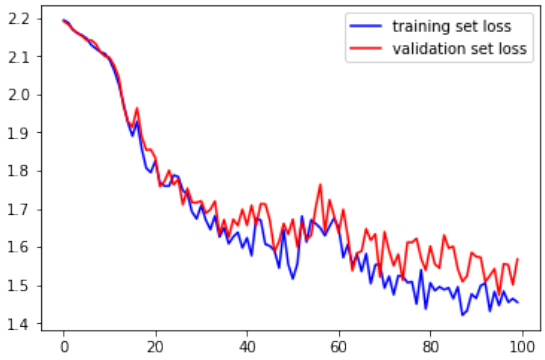
(1) Learning rate = 0.0030

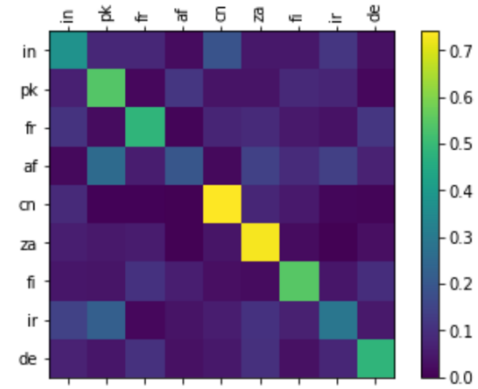




Macro Accuracy: 0.111111
Micro Accuracy: 0.106500

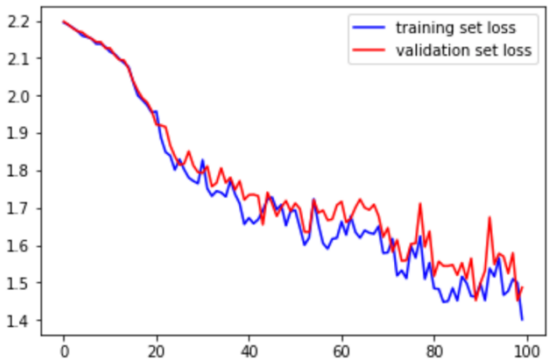
(2) Learning rate = 0.0020

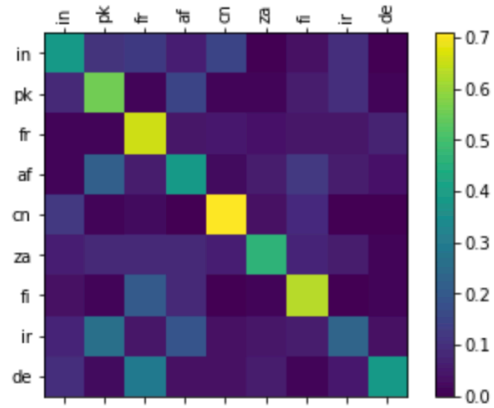




Macro Accuracy: 0.488409
Micro Accuracy: 0.487100

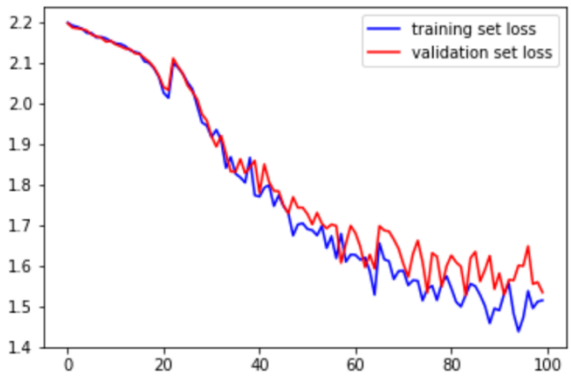
(3) Learning rate = 0.0015

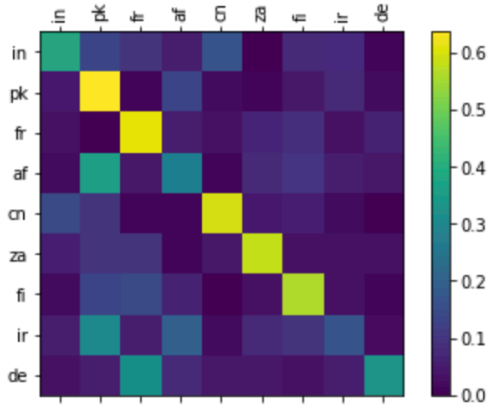




Macro Accuracy: 0.488864
Micro Accuracy: 0.487000

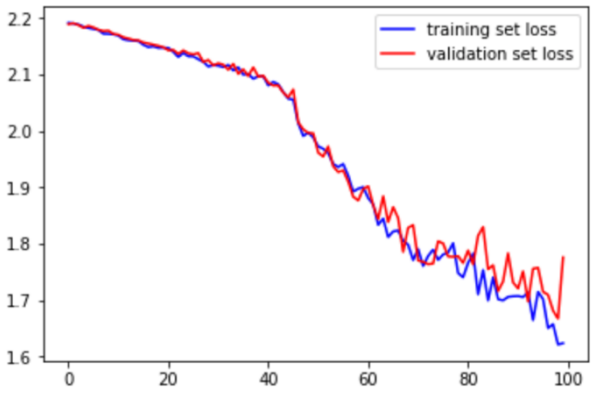
(4) Learning rate = 0.0010

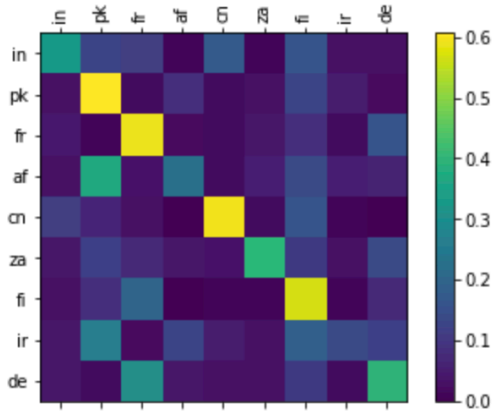




Macro Accuracy: 0.460937
Micro Accuracy: 0.460100

(5) Learning rate = 0.0005





Macro Accuracy: 0.429025
Micro Accuracy: 0.430900

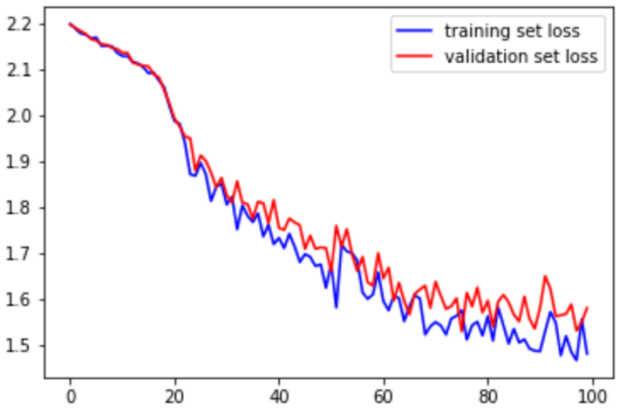
From the above graphs, we can notice that:

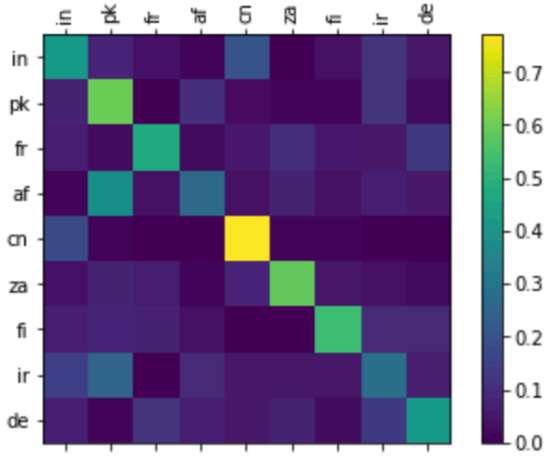
(a) If learning rate is too high (say higher than 0.002, like the 0.003 in the first case), overshoot will occur and there will be NaNs during training, so we have a very bad confusion matrix and accuracy at learning rate = 0.003
(b) By comparing the curves in loss graphs, we notice that as learning rate gets smaller, the loss decreases more slowly. Since each step is small, we need more steps of training to reduce loss.
(c) We can't guarantee that the smaller the learning rate is the higher accuracy we can get. In the above 5 cases, **learning rate = 0.0015** give us best accuracy.

4. **Experiment with the size of the hidden layer or the model architecture (at least 5 different sizes and/or modifications). How does this affect validation accuracy?**

   By keeping all other parameters the same (learning_rate = 0.0015, n_epochs = 100000, print_every = 5000, plot_every = 1000, etc), I try 5 different numbers of hidden nodes, n_hidden = 64, 90, 128, 200, and 256.

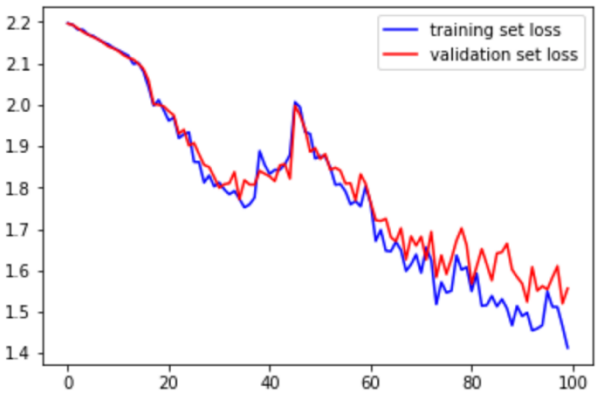
   (a) n_hidden = 64
   running time = 2m 51s

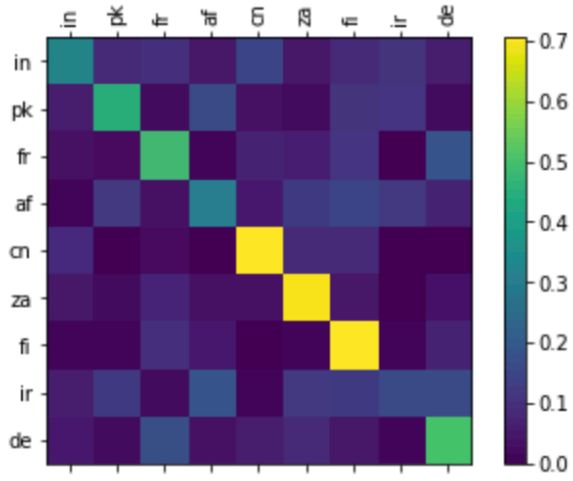




Macro Accuracy: 0.485127
Micro Accuracy: 0.487400

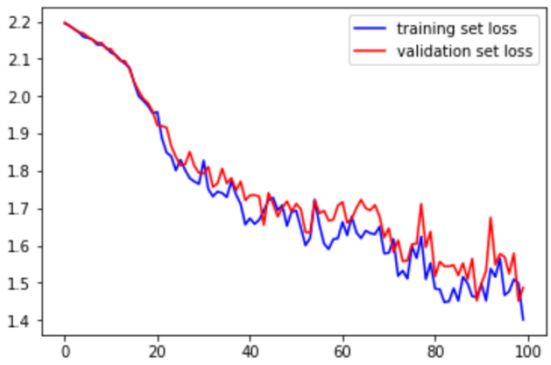
   (b) n_hidden = 90
   running time = 4m 54s

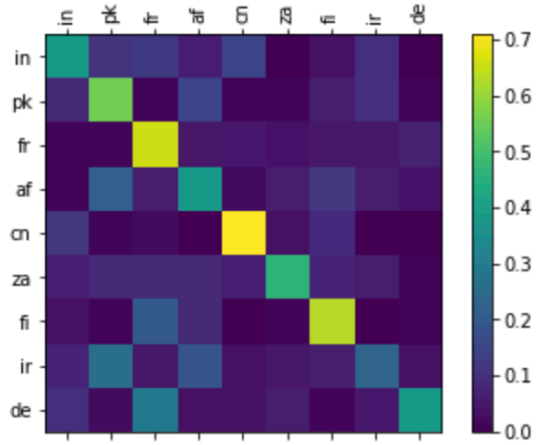




Macro Accuracy: 0.480810
Micro Accuracy: 0.481600

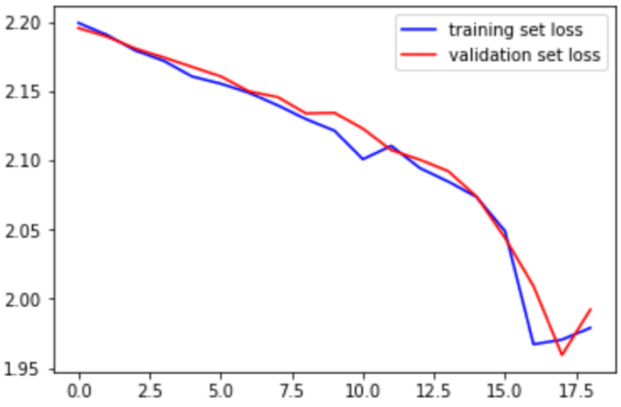
(c) n_hidden = 128
running time = 5m 5s

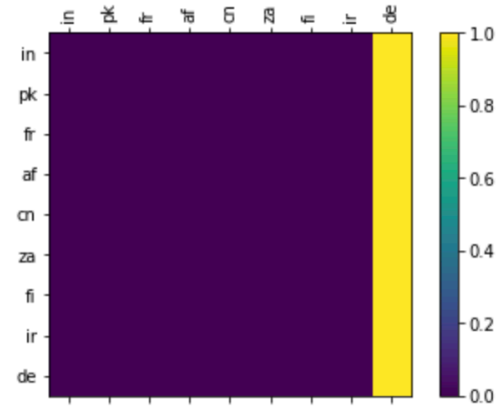




Macro Accuracy: 0.488864
Micro Accuracy: 0.487000

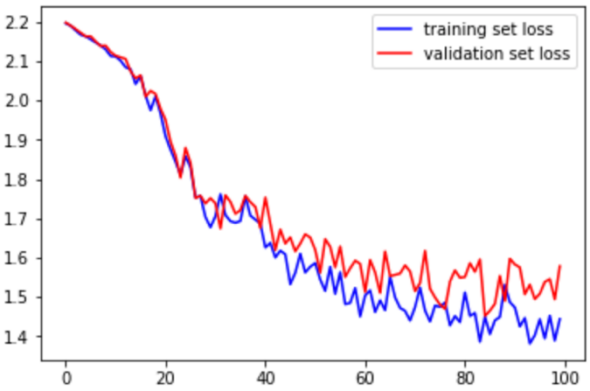
(d) n_hidden = 200
running time = 5m 49s

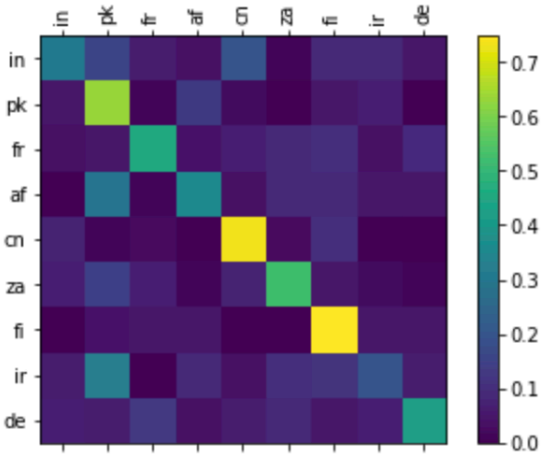




Macro Accuracy: 0.111111
Micro Accuracy: 0.113100

(e) n_hidden = 256
running time = 6m 14s





Macro Accuracy: 0.486945
Micro Accuracy: 0.486800

From the above graphs, we can notice that:

(a) More hidden nodes need more running time for training.
(b) We can't guarantee that the more hidden nodes added, the higher accuracy and better model we can get.
(c) When n_hidden = 90, losses suddenly increase and then decrease again.
(d) When n_hidden = 200, NaNs occur during training and the model fails.
(e) When n_hidden = 256, the train loss and validation loss seem to diverge. This may imply that too many hidden nodes could cause an overfitting.
(f) We'd better choose the best n_hidden case in case, here the best one is **n_hidden = 128.**